Luis Valderrama

08/28/2021

IT FDN 110 A

The Final

[GitHub Repository](https://github.com/lvalderr/IntroToProg-Python-Final) The Final

Working with Modules

# Intro

During the last assignments, I learned the concepts of working with Modules. This document covers the steps to create a script named, *Main.py*, designed to prompt the user to select from a four-option menu and execute the program based on the selection. This script introduces the concept of classes. The script is created using PyCharm as the Integrated Development Environment (IDE) and runs in PyCharm and Command Prompt.

# Class Module Set UP

In this section we examine the initial set up of the Class Modules that have been set up in three separate modules that will be linked and interact with the main script. These are: Data, IO and Processing Class.

## Data Class

This assignment uses two Data Classes, Person and Employee.

The **Person Class** is set up in this assignment to store the data about persons. The data in the class is defined with the **Constructor (and attributes)** which is defined as **def \_\_int\_\_** with variables relating back to the fields. In this case the variables for the constructor are **self, first\_name** and **last\_price.** The class also includes **Properties** that act as “getter” and “setters”. The “getters” are set as **@property** for **first\_name** and **last\_name,** while the “setters” are set as **@first\_name.setter** and **@last\_name.setter** and defined as **first\_name** and **last\_name**.Another pattern used in this class is the **Method** which defined by **def to\_string(self)** with the **\_\_str\_\_(self)** method to return the class’s data as strings. (Figure 1.1.a.)

The **Employee Class** is set up in this assignment to store the data about the employees. The data in the class is defined with the **Constructor (and attributes)** which is defined as **def \_\_int\_\_** with variables relating back to the fields. In this case the variables for the constructor are **self, employee\_id, first\_name** and **last\_price.** The class also includes one **Property** that act as “getter” and “setter”. The “getter” is set as **@property** for **employee\_id,** while the “setter” is set as **@employee\_id.setter**. Another pattern used in this class is the **Method** which defined by **def to\_string(self)** with the **\_\_str\_\_(self)**. (Figure 1.1.b.)

Text

Description automatically generated

***Figure 1.1.a: View of Person Data Class***

Text

Description automatically generated

***Figure 1.1.b: View of Employee Data Class***

## IO Class

This assignment uses four additional functions as outlined below in (Figure 1.2). Each function will be used for a specific presentation process in the Menu of Options and will be called based on the user’s input. A **class IO** (input/output) is set up to group the functions, variables and constants using **@staticmethod.** These functions are intended to take input from the user and display prompts and results.

Text

Description automatically generated

***Figure 1.2: View of IO Class***

## Processing Class

This assignment uses two processing functions, **save\_data\_to\_file** with the variables, **file\_name** which represents the text file that will be used with the program and the **List\_of\_rows** which represents the list where the data is stored. The other function is **read\_data\_from\_file** with the variable **file\_name.** Both functions use **@staticmethod** as outlined below in (Figure 1.3). Each function will be used for a specific process in the Menu of Options and will be called based upon the user’s input.

Text

Description automatically generated

***Figure 1.3: View of Processing Class***

# Main.py Program Set Up

In this section we examine the initial set up of the Main script which interacts with the Class Modules.

## Script Header and Objective

As mentioned in prior documents, the first part of the script outlines not only the title, description, date, and change log. But it also provides the reader with a high-level objective of the script and what it is designed to do. (Figure 2.1)

# --------------------------------------------------------------------------------------------#  
# Title: Main Module  
# Description: A module for testing  
# Change Log: (Who, When, What)  
# LValderrama, 8.27.2021, created Main Module  
# LValderrama, 8.28.2021, added imported processing and IO modules, while loop.  
# --------------------------------------------------------------------------------------------#  
  
# Objective:  
# 1. The Assignment09.py script is designed to:  
# a. Present a menu of choices for the user to select from.  
# b. Execute the program based on the choice made by the user.  
# c. Use Try/Catch Error Handling.  
# d. Use Class objects.  
# e. Work with other modules  
  
# Pseudo-Code:  
# Data -------------------------------------------------------------------------------------- #  
# Step 1: Declare variables

***Figure 2.1.: View of Script Header and Objective***

## Pseudo-Code

Before developing the script, the steps are outlined in the form of Pseudo-Code (Figure 2.2) to help translate the objective into the programing code and develop a usable program. In this example, there are 4 core steps that may expand into sub-steps as the script develops.

# Pseudo-Code:  
# Data -------------------------------------------------------------------------------------- #  
# Step 1: Declare variables

# Step 2: Import Module --------------------------------------------------------------------- #

# Main Body of Script ----------------------------------------------------------------------- #  
# Step 3: Load data from file into a list of employee objects when script starts ------------ #

# Step 4: Calling the functions from DataClasses, IOClasses, ProcessingClasses -------------- #

# Show user a menu of options --------------------------------------------------------------- #

# Get user's menu option choice ------------------------------------------------------------- #

# Show user current data in the list of employee objects ------------------------------------ #

# Let user add data to the list of employee objects ----------------------------------------- #

# Let user save current data to file -------------------------------------------------------- #

# Let user exit program --------------------------------------------------------------------- #

***Figure 2.2.: View of Script Header and Objective***

## Declaring Variables and Constants

As best practice, the variables and constants are declared before inserting the codes. (Figure 2.3)

strChoice = "" # Captures the user option selection  
strFileName = 'EmployeeData.txt' # The name of the data file  
lstTable = [] # A list that acts as a 'table' of rows

***Figure 2.3.: View of Declaring Variables***

## Importing Modules

The functions separated and organized in other scripts are imported or linked to the Main module using the import function which includes the name of the script, the class, and an alias. (Figure 2.4)

if \_\_name\_\_ == "\_\_main\_\_":  
 from DataClasses import Employee as Emp  
 from ProcessingClasses import FileProcessor as Fp  
 from IOClasses import EmployeeIO as Eio  
else:  
 raise Exception("There is was an error while importing the scripts")

***Figure 2.4.: View of Importing Modules***

## Loading Data from .txt File

When the program starts it opens the **EmployeeData.txt** and loads the three-column content into a list held in a table named **lstTable** as shown in (Figure 2.5.).

The function is defined using **def** followed by the name of the function, in this case **read\_data\_from\_file** and the variable given is **strFileName** which represents the text file **EmployeData.txt** where the list of Employees is saved. The parameters are entered as well as a return which in this example is the list of items. The actual function code is entered to open the .txt file, retrieve the data, and append it to the **lstTable**. The .txt file is then closed.

lstFileData = Fp.read\_data\_from\_file(strFileName)  
lstTable.clear()  
for line in lstFileData:  
 lstTable.append(Emp(line[0], line[1], line[2].strip()))

***Figure 2.5.: View of Loading Data from File to the List***

## Calling the Functions

In this section we examine the functions as they are called in the menu of options.

## Display Menu of Options

The function is called when the program starts running and it is encased in a **while(True)** loop as shown in (Figure 2.6.1.). The program will continue to loop through until the break is reached.

while True:  
 try:  
 # Show user a menu of options ----------------------------------------------------- #  
 Eio.print\_menu\_items()

***Figure 2.6.1.a: View of Displaying Menu of Options***

The actual function is in the IOClasses module as shown below in (Figure 2.6.1.b.) and is imported to the Main module using the import function as shown previously in (Figure 2.4).

@staticmethod  
def print\_menu\_items():  
 *""" Print a menu of choices to the user """* print('''  
 Menu of Options  
 1) Show current employee data  
 2) Add new employee data.  
 3) Save employee data to File  
 4) Exit program  
 ''')  
 print() # Add an extra line for looks

***Figure 2.6.1.b.: View of Menu of Option in the IOClass script.***

Displayed below in (Figure2.6.1.b.) is the program as it starts running in PyCharm and Command Prompt.

Text

Description automatically generated

***Figure 2.6.1.c: View of Menu of Option in the IOClass script.***

## Selecting Menu Option

The function is called as shown in (Figure 2.6.2.a) by way of entering a selection between 1 – 4.

strChoice = Eio.input\_menu\_options()

***Figure 2.6.2.a.: View of Selecting Menu Option***

The actual function is in the IOClasses module as shown below in (Figure 2.6.1.b.) and is imported to the Main module using the import function as shown previously in (Figure 2.4).

@staticmethod  
def input\_menu\_options():  
 *""" Gets the menu choice from a user* ***:return****: string  
 """* choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()  
 print() # Add an extra line for looks  
 return choice

***Figure 2.6.2.b.: View of option selection function***

## Show Current Employee Data (Option 1)

The function is called when the user enters “1” in the **input\_menu\_choice**. (Figure 2.6.3.a.)

if strChoice == '1':  
 Eio.print\_current\_list\_items(lstTable)  
 continue

***Figure 2.6.3.a.: View of Show Current Employee Data (Menu Option 1)***

The actual function is in the IOClass is defined as **print\_current\_list\_items** with the variable, **list\_of\_rows,** which is used to hold the data in the form of a three-column list. The function prints the data in the form of a list with values. The **print()** is set up to “unpack” the data. (Figure 2.6.3.b.)

@staticmethod  
def print\_current\_list\_items(list\_of\_rows: list):  
 *""" Print the current items in the list of Employee rows* ***:param*** *list\_of\_rows: (list) of rows you want to display  
 """* print("\*\*\*\*\*\*\* The current items employees are: \*\*\*\*\*\*\*")  
 for row in list\_of\_rows:  
 print(str(row.employee\_id) + "," + row.first\_name + "," + row.last\_name)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks

***Figure 2.6.3.b.: View of show current employee data from IOClass (Menu Option 1)***

Displayed below in (Figure2.6.3.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.6.3.c.: View of menu option 1 displayed in PyCharm and Command Prompt***

## Add New Employee Data (Option 2)

The function is called when the user enters “2” in the **input\_menu\_choice**. (Figure 2.6.4.a.)

elif strChoice == '2':  
 newEmployee = Eio.input\_employee\_data()  
 lstTable.append(newEmployee)  
 Eio.print\_current\_list\_items(lstTable)  
 continue

***Figure 2.6.4.a.: View of Add New Employee Data (Menu Option 2)***

The data processing function is in the IOClass module and is defined as **input\_employee\_data**. The function itself gets the employee ID, first and last name entered by the user as shown in (Figure 2.6.4.b.). The new data is appended to the list when the function is called in the Main module.

@staticmethod  
def input\_employee\_data():  
 *""" Gets data for a employee object* ***:return****: (employee) object with input data  
 """* global emp  
 try:  
 employee\_id = (input("What is the employee Id? - ").strip())  
 first\_name = str(input("What is the employee First Name? - ").strip())  
 last\_name = str(input("What is the employee Last Name? - ").strip())  
 print() # Add an extra line for looks  
 emp = DC.Employee(employee\_id, first\_name, last\_name)  
 except Exception as e:  
 print(e)  
 return emp

***Figure 2.6.4.b.: View of input employee data in IOClass (Menu Option 2)***

Displayed below in (Figure2.6.4.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.6.4.c.: View of menu option 2 displayed in PyCharm and Command Prompt***

## Save Employee Data to File (Option 3)

The function is called when the user enters “3” in the **input\_menu\_choice**. (Figure 2.6.5.a.)

elif strChoice == '3':  
 print('\n Would you like to save your data?')  
 strSaveToFileInput = input("Enter 'y' or 'n': ")  
 if strSaveToFileInput == 'n':  
 print('Data not saved!')  
 if strSaveToFileInput == 'y':  
 Fp.save\_data\_to\_file(strFileName, lstTable)  
 print('\nYour data is saved to', strFileName)  
 continue

***Figure 2.6.5.a.: View of save employee data (Menu Option 3)***

The processing function is in the ProcessingClass module and is defined as **save\_data\_to\_file** with the variable, **file\_name** and **list\_of\_rows** in the form of string. The function itself opens the EmployeeData.txt and saves the data to the file. (Figure 2.6.5.b.)

@staticmethod  
def save\_data\_to\_file(file\_name: str, list\_of\_rows: list):  
 *""" Write data to a file from a list of object rows* ***:param*** *file\_name: (string) with name of file* ***:param*** *list\_of\_rows: (list) of objects data saved to file* ***:return****: (bool) with status of success status  
 """* success\_status = False  
 try:  
 file = open(file\_name, "w")  
 for row in list\_of\_rows:  
 file.write(row.\_\_str\_\_() + "\n")  
 file.close()  
 success\_status = True  
 except Exception as e:  
 print("There was a general error!")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
 return success\_status

***Figure 2.6.5.b.: View of save employee data from ProccessingClasses module (Menu Option 3)***

Displayed below in (Figure2.6.5.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.6.5.c.: View of menu option 3 displayed in PyCharm and Command Prompt***

To view the saved data simply open the EmployeeData.txt. (Figure 2.6.5.d)

A screenshot of a computer

Description automatically generated

***Figure 2.6.5.d: View of the EmployeeData.txt content***

## Exit Program (Option 4)

There is no processing or IO functions associated with option 4. The program simply ends when the user enters 4 and the **break** in the **While(True)** loop is called. (Figure 2.6.6.a.)

elif strChoice == '4':  
 print("Goodbye!")  
 EndProgram = input('\n(Press Enter to End Program)')  
 break  
 else:  
 print('\nInvalid entry. Please enter a number from 1 to 4')  
except ValueError:  
 print('\nInvalid entry. Please enter a number from 1 to 4')

***Figure 2.6.6.a.: View of Exit the Program (Menu Option 4)***

Displayed below in (Figure2.6.6.b.) is the program as it runs in PyCharm and in Command Prompt after the loop break is called when the user enters 4.

Text

Description automatically generated

***Figure 2.6.6.b: Display Option 4 as it runs in PyCharm and Command Prompt***

# Summary

To recap, the last assignments introduced me to working with Modules. The example created for this assignment, *Main.py* is the result of steps taken to develop a script designed to prompt the user to select from a four-option menu and execute the program based on the selection. The program successfully allows the user to enter, display, read, and save the data in a .txt file. The program runs in PyCharm and Command Prompt and considers the concepts and best practices learned in this module.